Modeling Timing Requirements in Problem Frames Using CCSL

Xiaohong Chen*‡, Jing Liu*, *Frédéric* Mallet † and Zhi Jin‡
*Shanghai Key Laboratory of Trustworthy Computing, East China Normal University
†*Université Nice-Sophia Antipolis, INRIA Sophia Antipolis Méditerranée*
‡Key Laboratory of High Confidence Software Technologies, Ministry of Education, Peking University*

***Abstract*—As the embedded systems are becoming more and more complex, requirements engineering approaches are needed for modeling requirements, especially the timing requirements. Among various requirements engineering approaches, the Problem Frames(PF) approach is particularly useful in requirements modeling for the embedded systems due to the characteristic that PF pays special attention to the environment entities that will interact with the to-be software. However, no concern is given on timing requirements of PF at present. This paper studies how to add timing constraints on problem domains in PF. Our approach is to integrate the problem representation frame in PF with the timing representation mechanism of MARTE(Modeling and Analysis of Real Time and Embedded systems). A unified problem frame modeling process integrated with timing constraints is provided, and problem frame requirements with timing constraints expressed by MARTE/CCSL(Clock Constraint Specification Language) and clock construction operators are obtained.**

***Keywords*-requirements engineering; timing requirements; Problem Frames approach; CCSL; embedded systems;**

I. Introduction

Timing requirements are especially important in the embedded systems. For example, in the signalling system for high-speed trains, timing requirements are specified for avoiding train collision. In this kind of systems, timing requirements are critical for preventing damages and protecting the lives of people. This is immediately clear for all applications in the transport sector including computer controlled cars, trains and planes.

Addressing this problem, the MARTE [1](Modeling and Analysis of Real Time and Embedded systems) has recently been adopted by the OMG as a standard modeling language for real-time and embedded applications. It defines a broadly expressive Time Model that embodies a generic timed interpretation of UML models. Its notion of time covers both physical and logical times which is modeled by multiform time. Besides, many prevalence requirements engineering approaches start to pay attention to the timing requirements. For instance, the goal-oriented approaches [2], which take the system goals as the source of the requirements, capture the timing requirements with the typed first-order real-time logic. The agent oriented approaches [3], which use intentional actors as main clue to identify the requirements, also use the first-order logic to specify the timing requirements.

The Problem Frames (PF) approach [4] is a new and promising requirements engineering approach for describing, analyzing, and classifying software problems. It emphasizes that requirements exist in the environment of the to-be software, i.e. the problem domains that will interact with the to-be software. Therefore, the PF concentrates on the descriptions of the entities that will interact with the to-be software, and interactions between the entities and the to-be software. Compared with the goal oriented approaches and the agent oriented approaches, the PF approach is distinguished for its environment perspective. It is especially useful in embedded systems due to the above characteristics. However, no concern is given on the timing requirements in PF. A timing requirements modeling mechanism is needed.

This paper aims to study how to add timing constraints on problem domains in PF. Starting from modeling the time of a problem domain with a clock, this paper proposes to integrate the problem representation frame in PF with the timing representation mechanism of MARTE. In order to do this, a new icon, problem domain with clock which is formed by attaching a clock icon to the problem domain icon in the PF, is designed. A few clock construction operators in terms of problem domains are defined. At last, a unified problem frame modeling process integrated with timing constraints is provided, and problem frame requirements with timing constraints expressed by MARTE/CCSL(Clock Constraint Specification Language)[5] and clock construction operators are obtained.

In this paper, we focus on three kinds of timing requirements [5]:

- DelayRequirement that constrains the delay "from" a set of entities "until" another set of entities. It specifies the temporal distance between the execution of the earliest "from" entity and the latest "until" entity;
- RepetitionRate that defines the triggering period of an elementary Function;
- Input/outputSynchronization that expresses a timing requirement on the input/output synchronization of an Function.

The rest of this paper is organized as follows. Section II gives a brief introduction to the PF approach and MARTE/CCSL. Section III presents a timing conceptual model for PF with basic concepts in MARTE/CCSL. Section IV defines three clock construction operators in terms of

problem domains. Section V presents a unified requirements modeling process integrated with timing constraints, and shows the feasibility with an Anti-lock Braking System (ABS). Section VI presents some related work. Finally, Section VII concludes the paper.

II. Background

A. Problem Frames approach

The PF approach was first introduced into requirements engineering by Michael Jackson in 2001. In PF, software requirements are expressed as the expected changes on problem domains. These changes will be realized by building a software via the interaction between the software and the problem domains. Thus, in the PF approach, software requirements include identification of problem domains and interactions between the to-be software and the problem domains.

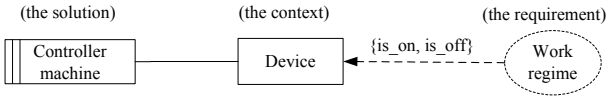

Figure 1. A simple problem diagram [6]

The result of the requirements description is a problem diagram. Fig. 1 gives a simple example of a problem diagram [6]. In the figure, the software *problem* is to specify a *machine* (the solution) to control a device (the problem context, consisting of a single *problem domain* in this case) so that a certain work regime (the *requirement*) is satisfied. The link between device and machine called interaction in the example indicates the *phenomena* that are shared between them. Phenomena can be, for instance, events or states or values. In this example, the shared phenomena are commands that the *controller machine* can issue to switch the device *on* and *off* (That such phenomena are controlled by the controller machine is indicated by a ! after the abbreviation CM). Phenomena *is_on* and *is_off* are the expected changes on the device. They are expected to happen between the machine and devices, so they are also a kind of interactions.

B. MARTE/CCSL

MARTE is a response to the OMG RFP to provide a UML Profile aiming at bringing in modeling software of the real-time and embedded domain. And CCSL is a non-normative language annexed to MARTE specification. As a declarative language that specifies constraints imposed on the clocks of a model, CCSL is widely used to support the specification of systems with multiple clock domains. In order to do this, it defines a set of elements required to support real-time and embedded domain. The following will give some definitions of the concepts that are used in this paper.

A clock is a model giving access to the time structure in MARTE[7]. Formally it is defined as a 5-tuple.

Definition 1: Clock

$$Clock \triangleq < I, \prec, D, \lambda, u >$$

where,

- I is a set of instants
- $\prec$ is a quasi-order relation on I, named strict precedence
- D is a set of labels
- $\lambda : I \to D$ is a labeling function
- u is a symbol, standing for a unit

For simplicity, this paper adopts a simplified definition of $Clock =< I, \prec >$.

Instants in clocks have relations. These relations can be unified defined as binary relation on set of instants:

$$IR : Instant \times Instant$$

Here three kinds of IR are distinguished:

- Precedence ($\preceq$): it represents causal dependency relation between instants. It has the following properties: (1) reflexive, that is $\forall a \in Instant$, we have $a \preceq a$, (2) transitive, that is $\forall a, b, c \in Instant$, if $a \preceq b$, $b \preceq c$, then we have $a \preceq c$.
- Coincidence ($\equiv \triangleq \preceq \cap \succeq$): it is a strong relation that forces simultaneous occurrences of instants. It has symmetric characteristic, that is $\forall a, b \in Instant$, if $a \equiv b$, then $b \equiv a$.
- Strict precedence ($\prec \triangleq \preceq \setminus \equiv$): it represents the sequential relation of occurrences of instants. It has transitive characteristic, that is $\forall a, b, c \in Instant$, if $a \prec b$, $b \prec c$, then we have $a \prec c$.

There are also a lot of operators for constructing new clocks using existing clocks. Here, we just list some operators that are related to this paper.

Definition 2: FilteredBy (▼)

$A \triangleq B \blacktriangledown \omega$, where A and B are discrete clocks, and ω is a binary word. A is constructed by B using

$\forall i \in I_A, \exists j \in I_B$

$$idex_A(i) = \omega \uparrow idex_B(j)$$

where I_A is the instants set of A, I_B is the instants set of B, $\omega \uparrow k$ is the index of the k^{th} 1 in ω.

This operator allows the selection of a subset of instants.

Definition 3: inf and sup

For clocks A and B, $C = inf(A, B)$ is the slowest clock faster than both A and B,

$\forall k \in N \backslash \{0\}$, $C[k]$=if $A[k] \preceq B[k]$ then$A[k]$ else $B[k]$

For clocks A and B, $D = sup(A, B)$ is the fastest clock among all clocks slower than both A and B,

$\forall k \in N \backslash \{0\}$, $D[k]$=if $A[k] \preceq B[k]$ then $B[k]$ else $A[k]$

where $A[k]$ means the k^{th} instant of A.

Definition 4: $a - b < t$

$a - b < t <=> b < a < b + t$ on sec, where a and b are instants, "+" is the delay operator and sec is another clock that discretize the IdealClock(can be seen as a real watch clock) each and every second.

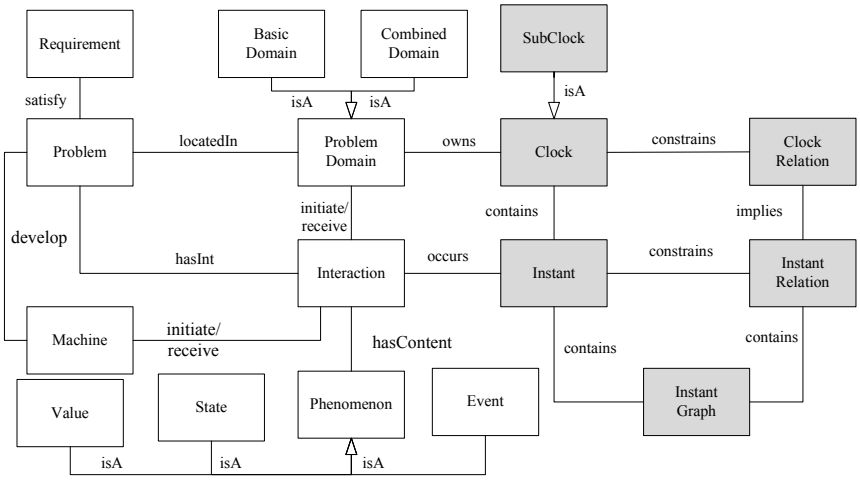

Figure 2. The timing PF conceptual model

III. A TIMING PF CONCEPTUAL MODEL

We have previously developed a conceptual model for describing software problems based on PF [8]. In this paper, we extend this conceptual model by associating the functionality related concepts with the time related concepts, so that a timing PF conceptual model is constructed. Fig. 2 shows the new conceptual model.

In Fig. 2, the white boxes and the links between them represent the concept categories and the associations between them based on the PF approach. They capture the main idea in the PF approach: a *Problem* is located in a set of real world *Problem Domains*, and is to develop a *Machine* to satisfy *Requirements*. A problem domain can be a *Basic Domain* or a *Combined Domain*. There are shared phenomena between the machine and the problem domains, i.e., *Interactions* between the Machine and the Problem Domains. An interaction has one initiator and one receiver that could be a machine or a problem domain. In fact, each interaction represents one individual action that the machine is involved in.

The gray boxes shows the time related concepts. These concepts are elicited from MARTE [9]. Time can be seen as a collection of *Clocks*. A clock may have many *subClocks*. Each clock specifies a totally ordered set of *Instants*. The relations between instants are called *Instant Relations*. To visually see the instants and instant relations, an *Instant Graph* is defined. Besides, there are relations between clocks which are called *Clock Relations*.

These two groups of concepts are related through the following relations. Each *problem domain* owns one *clock*. Interactions of each problem domain can be directly bound to time: the occurrences of *interactions* refer to time points, i.e., *instants* of related clocks.

Table I gives the exact meaning of the above concepts. Based on the extended timing model in Fig. 2, formal definitions of interaction and instant graph can be given. An interaction can be defined as a triple.

Table I
HIERARCHY OF THE CONCEPT CATEGORIES

Concept	Meaning
Problem	*a problem is a task to be accomplished by software development*
Machine	*the system to be built in a problem*
ProblemDomain	*a relevant real world entity that will interact with the machine*
BasicDomain	*a real world entity*
CombinedDomain	*entity that is composed by several basic domains*
Requirement	*functionalities which need to be satisfied*
Interaction	*an interaction is an observable phenomena shared by machine and a domain*
Phenomenon	*a phenomenon can be of state, value and event*
Event	*an event is an individual happening. Each event is indivisible and instantaneous*
Value	*a value is an intangible individual that is not subject to change*
State	*a state is a relation among causal domains and values. It can change over time*
Clock	*a clock is a virtual instrument used to indicate, keep, and coordinate time*
subClock	*a clock coordinates with another clock*
Instant	*the time point that interaction occurs*
Instant Relation	*the order relation between instants*
Clock Relation	*the relation between clocks*
Instant Graph	*a directed graph whose nodes are instants and the edges express the relations between two instants*

Definition 5: Interaction

$$Interaction \triangleq < initiator, receiver, content >$$

where,

- $initiator$ is the problem domain or machine that initiates the phenomenon
- $receiver$ is the problem domain or machine that receives the phenomenon
- $content$ is the shared phenomenon

An instant graph can be defined as 2-tuple.

Definition 6: Instant Graph

$$InstantGraph \triangleq < Ins, Rels >$$

where,

- Ins is a set of interactions
- $Rels$ is the set of instant relations between the elements in Ins

The instant relation can be precedence, coincidence or strict precedence as introduced in section II. These relations and their notations are shown in Fig. 3 [7].

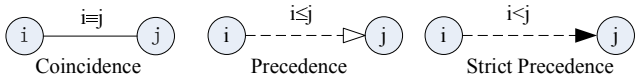

Figure 3. Legend of instant graph

IV. Clock construction operators

This section intends to define some clock construction operators. MARTE/CCSL defined many clock operators. But none of them concern problem domains. As the clocks within our model concern the problem domains, the original clock construction operators in CCSL need to be extended. Some new operators needs to be defined here.

Considering the problem domains in PF, clock construction operators can be defined in two ways. One way is for the same problem domain. In this situation, a new clock is defined by adding new instants to existing clock of the same problem domain. The other way is for the problem domains combination. That is, the existing problem domains may be too trivial, but the combination of these domains need to be modeled. After new problem domain being identified, the clock of combined domain is defined by combination of clocks of each problem domain.

Before we continue, some assumptions need to be made. A clock is either dense or discrete in MARTE. This paper only deals with discrete clocks. A discrete-time clock C for problem domain d can be denoted as $d.C$. It has a discrete set of instants, named I_C. Since I_C is discrete, it can be indexed by natural numbers in a fashion that respects the ordering on I_C: let $N^* = N \backslash \{0\}$, $idx : I_C \to N^*$, $\forall i \in I_C, idx(i) = k$ if and only if i is the k^{th} instant in I_C. Suppose $c[k]$ denotes the k^{th} instant in I_C(i.e., $k = idx_C(C[k])$). For any instant $i \in I_C$, $i-1$ is the unique immediate predecessor of i in I_C.

A. Operators for the same problem domain

Here we consider one situation when some interactions turn out to happen periodically. Then a new clock needs to be constructed by adding periodical instants to the old one. This operator is using FilteredBy operator in CCSL:

Definition 7: $d.A \triangleq d.B \blacktriangledown 0^o.(1.0^{p-1})^\omega$

where, d is a problem domain, p is the period, o is the offset, A is the new clock of d, B is the existing clock of d consisting in using a binary word with a single 1 in the periodic part. In this expression, for any bit b, b^0 stands for the empty binary word.

Intuitively, this means that many new instants are identified in clock A while repeating the same interactions in clock B. This is shown in Fig. 4.

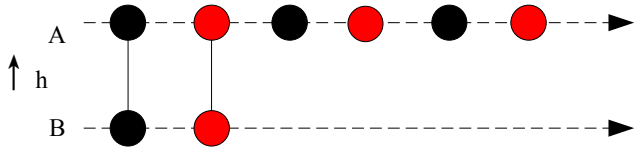

Figure 4. Instant graph for adding periodical interactions

B. Operators for the combined problem domains

The combination of problem domains can be classified into two kinds according to the domain structure. One kind is that domain d_1 and d_2 are sharing the same structure. For example, $sensor1$ and $sensor2$ of the same type are the same structured sensors. In terms of interactions, if they share the same phenomenon with the machine through interactions, they are the same structured domains, which is thus defined.

Definition 8: The same structured domains

Suppose $hasContent(X)$ returns the phenomenon that interaction X has. If for any interaction X of domain d_1, a corresponding interaction Y of domain d_2 can always be found such that

$hasContent(X) = hasContent(Y)$

and vice versa. Then d_1 and d_2 are the same structured domains.

In this situation, the combination of d_1 and d_2 acts like one domain d, and the clock of d can be defined by instants either in the clock of d_1 or clock of d_2. The instants of d will always be the slowest instants in d_1 and d_2. Thus, the construction operator of the same structured domain union can be defined as:

Definition 9: $d.A \triangleq d_1.B \oplus d_2.C$

Where, B is the discrete clock for domain d_1, and clock C is for domain d_2, d_1 and d_2 are the same structured domains, domain d is the union of d_1 and d_2, then the clock $d.A$ is:

$\exists hb : I_B \to I_A$, $\exists hc : I_C \to I_A$, such that

1) hb, hc is injective
2) hb, hc is order preserving:
   $(\forall i, j \in I_B)(i \prec_B j) \Rightarrow (hb(i) \prec_A h(j))$
   $(\forall i, j \in I_C)(i \prec_C j) \Rightarrow (hc(i) \prec_A h(j))$
3) an instant of I_B and its image are precedent:
   $(\forall i \in I_B) i \preceq h(i)$ $(\forall i \in I_C) i \preceq h(i)$
4) $\forall i_B[k] \in I_B$, $\forall i_C[k] \in I_C$
   $i_A[k] = sup(i_B[k], i_C[k])$

Intuitively, this means that each instant in clock A is the lowest instant happened in clock B and clock C. This can be seen in Fig. 5.

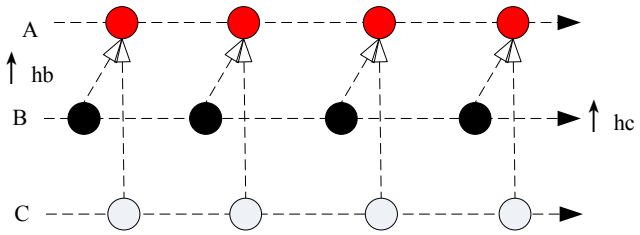

Figure 5. Instant graph for combining the same structured domains

The other situation is that the to-be combined domains are totaly different. Their interactions are quite different from each other. The clock of the union of these domains is the combination of clocks for basic domains by grouping instants happened in these domains together. Thus, the union operator for different problem domains can be defined:

Definition 10: $d.A \triangleq d_1.B \cup d_2.C$

Where, B is the discrete clock for domain d_1, and clock C for domain d_2, d_1 and d_2 are different domains, domain d is the union of d_1 and d_2, then the clock $d.A$ is:

$\exists hb : I_B \rightarrow I_A$, $\exists hc : I_C \rightarrow I_A$, such that

1) hb, hc is injective
2) hb, hc is order preserving:
   $(\forall i, j \in I_B)(i \prec_B j) \Rightarrow (hb(i) \prec_A h(j))$
   $(\forall i, j \in I_C)(i \prec_C j) \Rightarrow (hc(i) \prec_A h(j))$
3) an instant of I_B and its image are coincident:
   $(\forall i \in I_B) i \equiv h(i)$
   $(\forall i \in I_C) i \equiv h(i)$
4) $I_A = I_B \cup I_C$
   $\forall i_B[m] \in I_B$, $\forall i_C[n] \in I_C$, $i_B[m] \prec i_C[n]$
   $hb(i_B[m]) \prec_A hc(i_C[n])$

Intuitively, the instants of clock A are all the instants of clock B and C. It must be noticed that clock B and clock C are not asynchronous. This can be seen in Fig. 6.

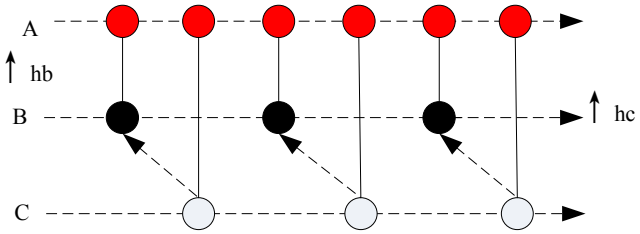

Figure 6. Instant graph for combining different domains

V. Timing Requirements Modeling Process: An Example

Fig. 7 gives a unified requirements modeling process by adding timing constraints to functional requirements obtained in PF. The input of this process is a problem diagram which includes problem domains, interactions etc.. It can be obtained by following process in [10]. The output is

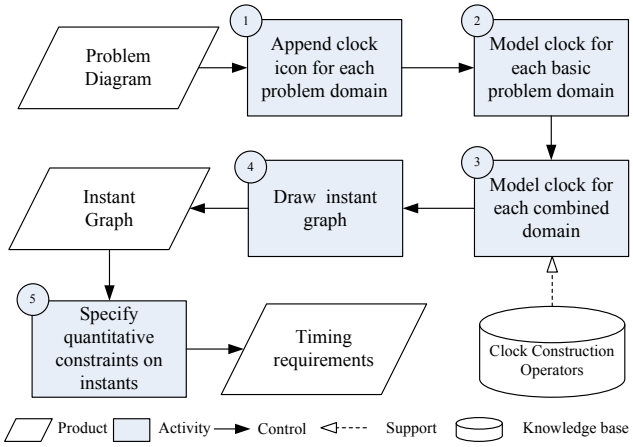

Figure 7. Timing requirements modeling process

timing requirements full of clock expressions in CCSL and clock construction operators defined in section IV. In Fig. 7 there are five main steps to guide the timing requirements modeling. We will use an example to the illustrate these steps. This example and the associated timing requirements are adapted from the ATESST report on EAST-ADL timing model [11]. The problem statements are as follows.

The ABS architecture consists of four sensors, four actuators and an indicator of the vehicle speed. The sensors (ifl, ifr, irl, irr) measure the rotation speed of the vehicle wheels. The actuators (ofl, ofr, orr and orl) indicate the brake pressure to be applied on the wheels.

The execution of the ABS is triggered by R. The values of the four sensors involved in the ABS must arrive within some delay (InputSynchronization). A similar OuputSynchronization delay is represented on the actuators side. The delay from the first event on the input set of the ABS until the last event occurrence on the output set is also needed.

Example 1: The problem diagram of the ABS system is shown in Fig. 8. In this figure the combined domain *Sensor* has four basic domains: *ifl, ifr, irl* and *irr*. Similarly, the combined domain *Actuator* has four basic domains: *ofl, ofr, orl*, and *orr*. The interactions between *ABS* and *Sensor*, *ABS* and *Actuator* are asserted as follows:

$int_1 =< ABS, Sensor, R >$
$int_2 =< Sensor, ABS, rotation\ speed >$
$int_3 =< ABS, Actuator, break\ pressure >$
$int_4 =< Actuator, ABS, speed >$

Obviously, *ofl, ofr, orl*, *orr* and *Sensor* are the same structured domains, and *ofl, ofr, orl*, *orr* and *Actuator* are the same structured domains. For simplicity, this paper just uses int_{ij} (i can be 1, 2, 3, or 4, and j is the name of the domain) to denote their interactions.

Step 1: append a clock icon to each problem domain

This step is to declare clocks for problem domains. A particular way is to append a clock icon to each domain

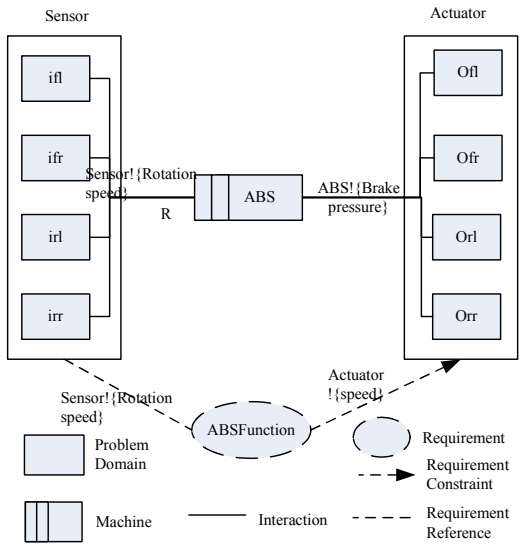

Figure 8. Problem diagram of ABS

icon in the problem diagram.

Example 2: Back to our example, append a clock to each problem domain in Fig. 8. Therefore Fig. 9 is obtained, and the following assertions can be got from Fig. 9:

Clock $Senser.C_{Sensor}, Actuator.C_{Actuator}$;
Clock $ifl.C_{ifl}, ifr.C_{ifr}, irl.C_{irl}, irr.C_{irr}$;
Clock $ofl.C_{ofl}, ofr.C_{ofr}, orl.C_{orl}, orr.C_{orr}$;

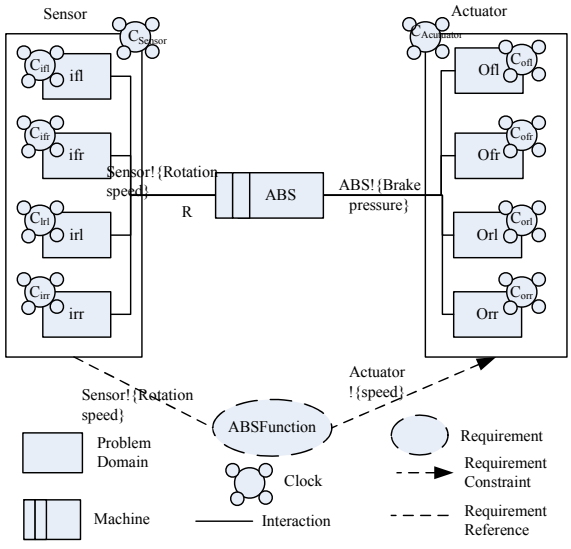

Figure 9. Problem diagram of ABS with clocks

Step 2: model clocks for basic domains

This step can be finished in two sub-steps:

1) find instants for each clock

The instants for each domain are the occurrences of interactions that related domain initiates or receives. So this step is to identify the occurrence of each interaction.

2) specify strict precedence of instant relations within each clock

This step is to specify the relations between instants.

Example 3: The basic domains in this example are sensors ifl, ifr, irl, irr and actuators ofl, ofr, orl, orr. The moments that int_{1ifl} and int_{2ifl} happened are the instants of C_{ifl}, recording as $O(int_{1ifl})$, $O(int_{2ifl})$, then $I_{ifl} = \{O(int_{1ifl}), O(int_{2ifl})\}$. Similarly we get:

$I_{ifl} = \{O(int_{1ifl}), O(int_{2ifl})\}$;
$I_{ifr} = \{O(int_{1ifr}), O(int_{2ifr})\}$;
$I_{irl} = \{O(int_{1irl}), O(int_{2irl})\}$;
$I_{irr} = \{O(int_{1irr}), O(int_{2irr})\}$;
$I_{ofl} = \{O(int_{3ofl}), O(int_{4ofl})\}$;
$I_{ofr} = \{O(int_{3ofr}), O(int_{4ofr})\}$;
$I_{orl} = \{O(int_{3orl}), O(int_{4orl})\}$;
$I_{orr} = \{O(int_{3orr}), O(int_{4orr})\}$;

As to the instants relation, for example, in clock C_{ifl}, int_{1ifl} must happens before int_{2ifl}, so $O(int_{1ifl}) \prec O(int_{2ifl})$. Similarly, the other strict precedence relations can be obtained:

$O(int_{1ifl}) \prec O(int_{2ifl})$;
$O(int_{1ifr}) \prec O(int_{2ifr})$;
$O(int_{1irl}) \prec O(int_{2irl})$;
$O(int_{1irr}) \prec O(int_{2irr})$;
$O(int_{3ofl}) \prec O(int_{4ofl})$;
$O(int_{3ofr}) \prec O(int_{4ofr})$;
$O(int_{3orl}) \prec O(int_{4orl})$;
$O(int_{3orr}) \prec O(int_{4orr})$;

Step 3: model clocks for combined domains

This step is to construct clocks from existing clocks using combined clock construction operators defined in section IV. To do this efficiently, we'd better start by finding the combined domain and its basic domains.

Example 4: $Sensor$ is the combination of ifl, ifr, irl, and irr. Sensor ifl, ifr, irl and irr are the same structured domain, Thus, we have:

$Sensor.C_{Sensor}$=$ifl.C_{ifl} \oplus ifr.C_{ifr} \oplus irl.C_{irl} \oplus irr.C_{irr}$

Similarly, we get:

$Actuator.C_{Actuator}$= $C_{ofl} \oplus ofr.C_{ofr} \oplus orl.C_{orl} \oplus orr.C_{orr}$

Especially, the instant relations needs to be specified.

In clock C_{Sensor}, the moments that int_1 and int_2 happened are the instants of this clock. Recording them as $O(int_1)$, $O(int_2)$, then $I_{Sensor} = \{O(int_1), O(int_2)\}$. Considering the relations of $Sensor$, ifl, ifr, irl, and irr, the occurrence of int_1 should be the slowest among the occurrences of int_{1ifl},int_{1ifr},int_{1irl}, and int_{1irr}, thus we get:

O(int_1)=sup(O(int_{1ifl}),O(int_{1ifr}),O(int_{1irl}), O(int_{1irr}))

Similarly we get:

O(int_2)=sup(O(int_{2ifl}),O(int_{2ifr}),O(int_{2irl}), Oint_{2irr}))

In clock $C_{Actuator}$, $I_{Actuator} = \{O(int_3), O(int_4)\}$

Likewise, we get:

O(int_3)=sup(O(int_{3ofl}),O(int_{3ofr}),O(int_{3orl}), O(int_{3orr}))

O(int_4)=sup(O(int_{4ofl}),O(int_{4ofr}),O(int_{4orl}), O(int_{4orr}))

Step 4: draw the instant graphs

This step is to specify the instant relations within and among clocks visually. Usually we just specify the 3 relations: precedence, coincidence, and strict precedence. The instants relations identify needs close participation of requirement providers.

Example 5: For example, between C_{Sensor} and $C_{Actuator}$, int_2 happens before int_3, so $O(int_2) \prec O(int_3)$.

Between C_{Sensor} and C_{ifl}, int_1 happens no early than int_{1ifl}, so $O(int_{1ifl}) \preceq O(int_1)$.

Similarly, we get the other instant relations as well as instant relations within one clock in step 2 as shown in Fig. 10.

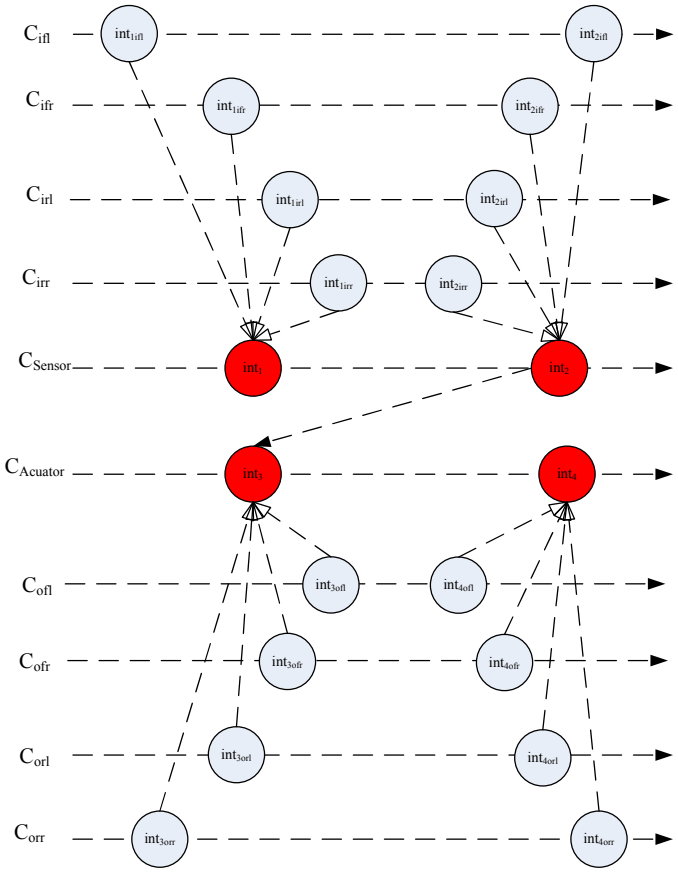

Figure 10. The instant graph for ABS

Step 5: specify quantitative constraints on instants

This step is to specify the quantitative constraints on instants. For example, intervals between instants is no more than 5 seconds. Of course, not all the instants need to do this. Only some important instants are required.

Example 6: The three kinds of quantitative requirements considered in this paper are as follows.

1) Repetition Rate

For example, this kind of requirements may be written as:

The ABS function must be executed every 5ms with a maximum jitter of 1ms.

This function will be implemented by the combination of *Sensor* and *Actuator* periodically. *Sensor* and *Actuator* are different domains, so the existing clock will be:

$Sensor.C_{Sensor} \cup Actuator.C_{Actuator}$

According to the periodic construction operator in section IV, the other parameters include the offset being 0 and the period being 5. Then the new clock C_{new} is constructed as: C_{new}=$(Sensor.C_{Sensor} \cup Actuator.C_{Actuator})\blacktriangledown(1.0^{5-1})^{\omega}$

2) Delay Requirement

This kind of requirements may be stated as:

At each iteration, the distance between the reception of the first input and the emission of the last output must be less than 3ms.

Take int_1 of $Sensor$ as an input, and int_4 of $Actuator$ as an output. Then the first input can be written as:
$O_{1inf} = inf(O(int_{1ifr}), O(int_{1ifl}), O(int_{1irl}), O(int_{1irr}))$

The emission of the last output is $O(int_4)$. And finally, the interval between the two can be:

$O(int_4) - O_{1inf} < 30$

3) Input & output synchronization

Input & output synchronization are specializations of a delay requirement. An input synchronization delay requirement for the Function ABS bounds the temporal distance between the earliest input and the latest input. For example, *an input synchronization of 0.5ms is needed.*

The first input is O_{1inf}, and the last input is $O(int_1)$. Then the input synchronization of 0.5ms is:

$O(int_1) - O_{1inf} < 5$

Likewise, an output synchronization delay requirement for the Function ABS bounds the temporal distance between the earliest output and the latest output. For example, *an output synchronization of 0.5ms is needed.*

The first output is:
$O_{4inf} = inf(O(int_{4ofr}), O(int_{4ofl}), O(int_{4orl}), O(int_{4orr}))$

The last output is $O(int_4)$.Then the output synchronization of 0.5ms is:

$O(int_4) - O_{4inf} < 5$

After the whole process, problem frame based requirements are obtained. They not only includes the functional requirements in the form of original problem diagram, but also the timing requirements in the form of problem domain clock icons and constraints through clock construction operators and CCSL.

VI. Related work

There are many efforts for modeling timing requirements. The goal oriented approaches [2] view goals as the source of requirements. One of the representative work is KAOS [2](Knowledge Acquisition in Automated Specification). The logic used in KAOS is typed first-order real-time logic. It consists of traditional temporal operators, together with additional real-time operators for specifying properties involving real-time deadlines. A key feature of the logic is

its ability to model real-time properties concisely without referring explicitly to a time variable.

The agent oriented approaches [3] use actor as a clue to identify requirements. Their representative work is i* framework [12] and Formal Tropos [13]. The Formal Tropos language offers all the primitive concepts of i*, but supplements them with a rich temporal specification language inspired by KAOS. Formal Tropos also uses a linear-time typed first-order temporal logic.

The other efforts of agent oriented have also been made to specify timing requirements. For example, ALBERT-II(Agent-Oriented Language for Building and Eliciting Real-Time Requirements) [14], a formal framework designed for specifying distributed real-time systems, is based on temporal logic. Agents' states and behavior are specified through constraints expressed in the logic-based notation.

However, all the above approaches are based on the logic. A limitation of the logic currently used is that the time domain is assumed to be discrete. This makes it difficult to accurately capture and reason about properties involving time derivatives and integrals of time-continuous variables. Compared with these approaches, our approach is based on the MARTE/CCSL, a multiform timed system which is more close to the real world than the other general multiple clock systems. The time domain combines the discrete and continuous characteristics by using instants as the discrete time points. Another advantage of our approach is that our timing requirements are deeply rooted in the functional requirements.

VII. Conclusion

Timing requirements are of great importance in the embedded systems. In this paper, we propose an approach to model the timing requirements based on the PF approach using CCSL. The main contribution includes:

- A new timing PF model is constructed. This model introduces clocks for problem domains into PF, which makes timing requirements description in PF possible.
- A method for specifying timing requirements in PF is provided. This method is based on the functional requirements description in PF, which makes solid foundation for timing requirements description. It is fulfilled with timing constraints proposed in MARTE/CCSL.

The timing requirements modeling results in timing problem frame requirements. Further work needs to be done for verifying timing requirements, and deriving machine specifications from the timing requirements. Besides, this paper only deals with three kinds of timing requirements. More timing requirements are under consideration.

Acknowledgment

The authors wish to thank Prof. Robert de Simone for his constructive comments and suggestions. This work was supported by the National Basic Research and Development 973 Program of China (Grant No.2009CB320702), and National Natural Science Foundation of China (Grant No.90718014, No.90818026, No.61021004).

References

[1] C. André, F. Mallet, and R. de Simone, "Modeling time(s)," *Lecture Notes in Computer Science*, vol. 4735, 2007.

[2] A. van Lamsweerde, "Formal refinement patterns for goal-driven requirements elaboration," in *Proceedings of the 4th ACM Symposium on the Foundations of Software Engineering (FSE4)*, San Francisco, USA, 1996, pp. 179–190.

[3] E.Yu, "Agent orientation as a modeling paradigm," *Wirtschaftsinformatik*, vol. 43, no. 2, pp. 123–132, 2001.

[4] M.Jackson, *Problem Frames: Analyzing and Structuring software development problems*. Harlow, England:Addison-Wesley, 2001.

[5] F. Mallet, M. Peraldi-Frati, and C. André, "Marte CCSL to execute East-ADL timing requirements," in *Int. Symp. on Object/component/service-oriented Real-time distributed Computing (ISORC'09)*. Japan, Tokyo: IEEE Computer Press, March 2009, pp. 249–253.

[6] K. Cox, J. G. Hall, and L. Rapanotti, "A roadmap of problem frames research," *Information and Software Technology*, vol. 47, no. 14, pp. 891–902, 2005.

[7] C. André and F. Mallet, "Specification and verification of time requirements with ccsl and esterel," in *Proceedings of the 2009 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems, LCTES 2009*, Dublin, Ireland, June 2009, pp. 167–176.

[8] Z.Jin, X.Chen, and D.Zowghi, "Performing projection in problem frames using scenarios," in *Proceedings of the 16th Asia-Pacific Software Engineering Conference(APSEC 2009)*, 2009, pp. 249–256.

[9] OMG, "Uml profile for modelling and analysis of real- time and embedded systems (marte)," http://www.omgmarte.org/.

[10] X. Chen and Z. Jin, "An ontology-guided process for developing problem frame specification: an example," in *Proceedings of the 3rd International Workshop on Applications and Advances of Problem Frames (IWAAPF 2008)*, 2008, pp. 36–39.

[11] R. Johansson, H. Lonn, and P. Frey, "Atesst timing model," ITEA, Tech. Rep., 2008, deliverable D2.1.3.

[12] E. Yu, "Modelling organizations for information systems requirements engineering," in *Proceedings of First IEEE Symposium on Requirements Engineering*, 1993, pp. 34–41.

[13] E.Yu, "Towards modeling and reasoning support for early-phase requirements engineering," in *Proceedings of the 3rd IEEE international Symposium on Requirements Engineering (RE'97)*. Washington DC: IEEE Computer Society, 1997, pp. 226–235.

[14] P. Bois, "The albert ii language - on the design and the use of a formal specification language for requirements analysis," Ph.D. dissertation, Dept. of Computer Science, University of Namur, Namur, Belgium, 1995.